Programmazione Orientata agli Oggetti in Linguaggio Java

Ruoli e Responsabilità: Incapsulamento

versione 2.1

G. Mecca – Università della Basilicata – mecca@unibas.it

Sommario

- Interfaccia e Implementazione
- Incapsulamento
- Utilizzo delle Liste
  - ⇨La Media Pesata
- Il Concetto di “interface”

Interfaccia e Implementazione

- Implementazione di un componente
  - ⇨codice sorgente di una classe
  - ⇨che descrive la natura e le capacità del componente
  - ⇨quali sono le sue proprietà
  - ⇨quali sono i suoi metodi e di quali operazioni si compongono
  - ⇨quali sono i livelli di visibilità

Interfaccia e Implementazione

- Ciclo di vita di un componente
  - ⇨due fasi principali
- Sviluppo
  - ⇨concepimento (progettazione)
  - ⇨scrittura e compilazione
- Utilizzo
  - ⇨esecuzione dei metodi all'interno di altri componenti

Interfaccia e Implementazione

- In ciascuna fase
  - ⇨intervengono attori diversi
- Sviluppo
  - ⇨sviluppatore del componente
- Utilizzo
  - ⇨sviluppatore dei componenti che utilizzano il componente in questione
  - ⇨o, nella nostra metafora: altri componenti che chiamano metodi del componente

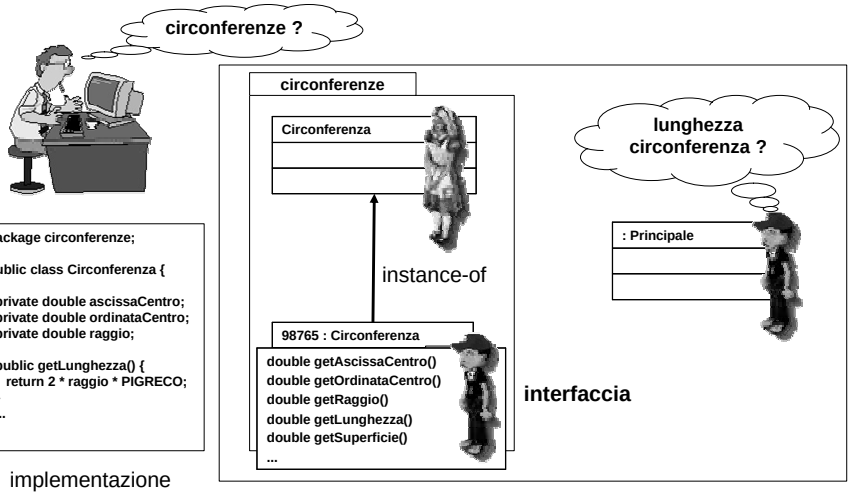
Interfaccia e Implementazione

- Un principio fondamentale
  - ⇨i diversi attori hanno prospettive diverse rispetto allo stesso componente
- Per lo sviluppatore del componente
  - ⇨sono rilevanti tutti i dettagli relativi all'implementazione del componente
  - ⇨tutto il codice sorgente della classe corrispondente

Interfaccia e Implementazione

- Per lo sviluppatore degli altri componenti
  - ⇨la prospettiva è completamente diversa
  - ⇨è rilevante solo l'interfaccia del componente
- Interfaccia del componente
  - ⇨l'elenco dei servizi che il componente è in grado di offrire all'esterno
  - ⇨e il modo per utilizzarli
  - ⇨in pratica: tutte le proprietà pubbliche ed i metodi pubblici del componente

Interfaccia e Implementazione



Interfaccia e Implementazione

- Metafora
  - ⇨interfaccia: quello che il robot sa fare >> i compiti che è disponibile a svolgere
  - ⇨quali sono e come è possibile richiederli
  - ⇨implementazione: tutti i dettagli di come lo fa >> gli strumenti e le tecniche per svolgere i suoi compiti
  - ⇨cosa conosce e quali passi esegue

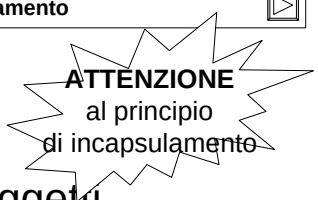
Interfaccia e Implementazione

- Analogo
  - ⇨supponiamo che il nostro omino sia un idraulico
  - ⇨alla massaia con la lavatrice rotta interessa la sua interfaccia, ovvero: “riparami la lavatrice”, “ecco fatto, sono 500 Euro”
  - ⇨non le interessa la sua implementazione (come e con quali strumenti ripara la lavatrice)

Interfaccia e Implementazione

- L'accoppiamento con altri componenti
  - ⇨è strettamente collegato all'interfaccia
  - ⇨gli altri componenti inviano messaggi riguardanti proprietà e metodi pubblici
  - ⇨cambiare l'interfaccia richiede di cambiare anche i componenti accoppiati
- Come scegliere l'interfaccia ?
  - ⇨in modo da minimizzare l'accoppiamento

Incapsulamento



- Nella programmazione a oggetti
  - ⇨questo corrisponde ad un principio essenziale
- Incapsulamento ("Information Hiding")
  - ⇨l'interfaccia di un componente deve essere la più semplice e piccola possibile
  - ⇨l'implementazione dovrebbe essere il più possibile nascosta
  - ⇨ovvero: usare public il meno possibile

Incapsulamento

- Alcune consuetudini generali
  - ⇨le proprietà di un componente dovrebbero essere parte dell'implementazione e NON dell'interfaccia >> private
  - ⇨tutti i metodi "di servizio" del componente – metodi non destinati ad essere chiamati dall'esterno NON dovrebbero far parte dell'interfaccia >> private

Incapsulamento

- Attenzione
  - ⇨il concetto di incapsulamento è un concetto generale della programmazione
  - ⇨vale anche per la programm. procedurale
- La grande differenza
  - ⇨nella programmazione procedurale è una disciplina: non è possibile nascondere realmente l'implementazione di una libreria
  - ⇨nella programmazione a oggetti è un vincolo

Utilizzo delle Liste

- Un esempio tipico
  - ⇨la libreria per gestire liste
- Tipicamente
  - ⇨un'unica interfaccia (inserisci, cancella, ...)
  - ⇨due principali implementazioni
  - ⇨implementazione statica (basata su array e indicatore di riempimento)
  - ⇨implementazione doppiamente collegata

Utilizzo delle Liste

- Nella programmazione procedurale
  - ⇨non è possibile impedire ai programmatori di utilizzare direttamente parti dell'implementazione
  - ⇨esempio: non è possibile nascondere completamente l'indicatore di riempimento
  - ⇨questo rende più difficile sostituire una versione della libreria con l'altra

Utilizzo delle Liste

- In Java
  - ⇨due versioni della lista
  - ⇨java.util.ArrayList: implementazione basata su array e indicatore di riempimento
  - ⇨java.util.LinkedList: implementazione doppiamente collegata
  - ⇨le classi hanno la stessa interfaccia
  - ⇨l'implementazione è rigorosamente nascosta
  - ⇨di conseguenza le classi sono sostituibili

Utilizzo delle Liste

- L'interfaccia comune

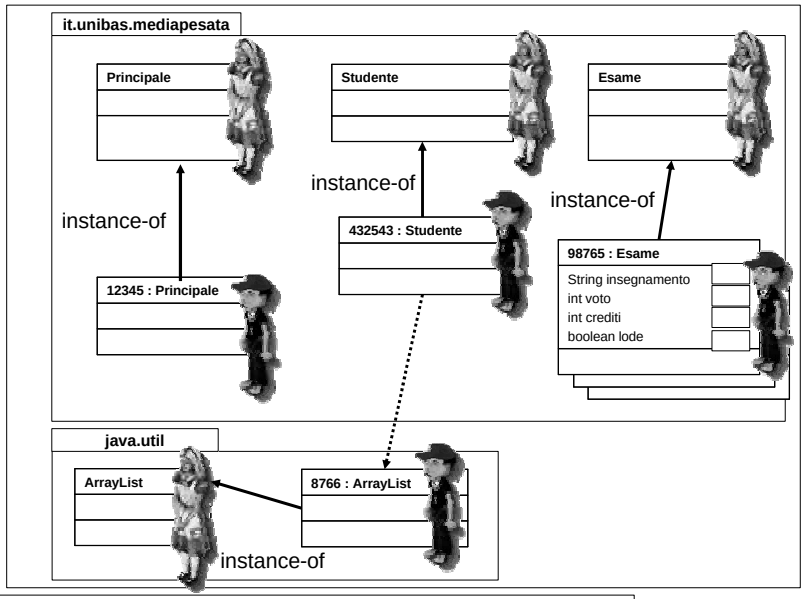
boolean add(Object o)	aggiunge in coda
void add(int index, Object element)	aggiunge in posizione
Object get(int index)	preleva l'elemento i-esimo
Object remove(int index)	elimina in posizione
Object set(int index, Object element)	cambia l'elemento i-esimo
int indexOf(Object o)	cerca l'elemento o
int size()	dimensione
boolean isEmpty()	lista vuota
void clear()	svuota la lista

Utilizzo delle Liste

- Una particolarità delle liste
  - ⇨le liste di java.util contengono esclusivamente riferimenti ad istanze della classe Object
- Attenzione
  - ⇨qui sono fortemente coinvolti ereditarietà e polimorfismo
  - ⇨per ora solo regole meccaniche
  - ⇨per illustrarle: la Media Pesata

Media Pesata

- Tre classi in it.unibas.mediapesata
  - ⇨Principale
  - ⇨Studente
  - ⇨Esame
- Inoltre
  - ⇨lo studente deve conoscere i suoi esami
  - ⇨una lista di esami universitari
  - ⇨java.util.ArrayList



Media Pesata

- Le operazioni sulla lista
  - ⇨aggiungi esame
  - ⇨fornisci i dati dell'esame i-esimo
  - ⇨fornisci il numero di esami
- Attenzione
  - ⇨la classe java.util.ArrayList rappresenta una lista di riferimenti ad oggetti della classe Object

Media Pesata

- Ricapitolazione veloce
  - ⇨tutte le classi di Java estendono implicitamente Object
- Regola n. 1: eredità delle caratteristiche
  - ⇨tutti gli oggetti ereditano i metodi di Object
- Regola n. 2: sostituibilità
  - ⇨tutti gli oggetti di Java possono essere considerati anche oggetti di tipo Object

Media Pesata

- Conseguenze della lista di Object
  - ⇨la lista può contenere riferimenti ad oggetti qualunque (qualunque oggetto può essere considerato un Object)
  - ⇨ma ci sono piccole conseguenze
  - ⇨anche in questo caso le vediamo come regole meccaniche

Media Pesata

- Regola n. 1: inserimento in una lista
  - ⇨il riferimento da inserire viene “trasformato” in un riferimento ad Object (es: rif. ad Esame)
  - ⇨non cambia il suo valore
- Regola n. 2: prelevamento dalla lista
  - ⇨il riferimento deve essere “ritrasformato” in un riferimento del tipo originale attraverso un cast (es: Esame)
  - ⇨anche in questo caso non cambia il valore

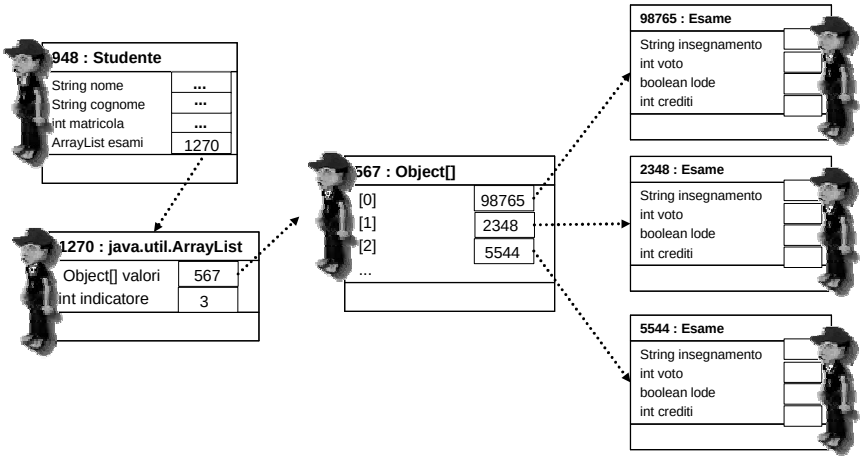
Media Pesata

```
package it.unibas.mediapesata;
public class Studente {
   private String nome;  private String cognome; private int matricola;
   private java.util.ArrayList esami = new java.util.ArrayList();

   public void addEsame(Esame esame) {
      this.esami.add(esame);
   }

   public Esame getEsame(int i) {
      return (Esame)this.esami.get(i);
   }

   public int getNumeroEsami() {
      return this.esami.size();
   }
```

il riferimento all’Esame viene trasformato in un rif. a Object: un oggetto di tipo Esame è anche un oggetto di tipo Object

il metodo get restituisce un riferimento ad un Object; il cast è necessario per trasformarlo in un riferimento ad un Esame

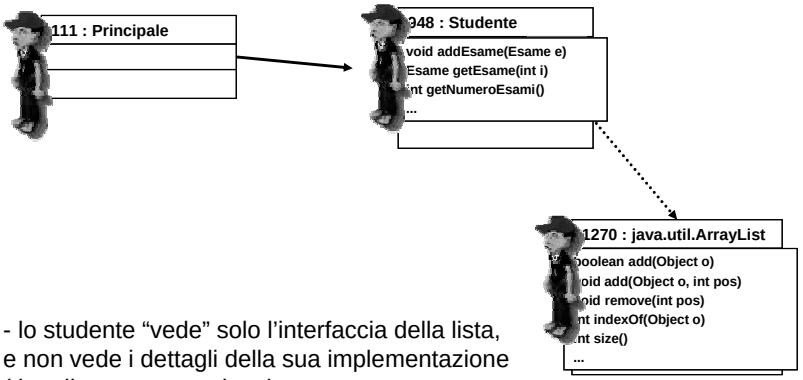
Media Pesata



Media Pesata

- In realtà però
  - ⇨ le implementazioni dei vari componenti sono nascoste al loro interno (incapsulamento)
- Infatti
  - ⇨ Studente si basa esclusivamente sull'interfaccia di java.util.ArrayList (add(), get(), size())
  - ⇨ Principale si basa esclusivamente sull'interfaccia di Studente per la gestione della lista di esami (addEsame(), getEsame(), getNumeroEsami())

Media Pesata



- lo studente “vede” solo l’interfaccia della lista, e non vede i dettagli della sua implementazione (tipo di rappresentazione)
- Principale vede solo l’interfaccia dello studente e non i dettagli dell’implementazione (tipo di lista)

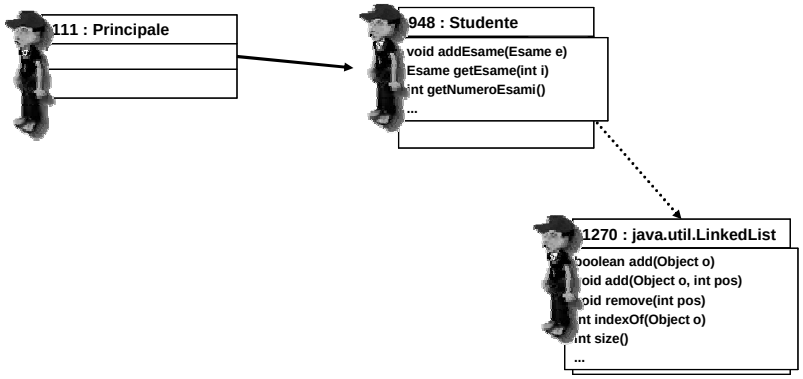
Media Pesata

>> Studente.java

- Vantaggi dell’incapsulamento
  - ⇨è possibile cambiare il tipo di lista
  - ⇨per esempio per ragioni di prestazioni
  - ⇨il resto dell’applicazione non è influenzata

```
package it.unibas.mediapesata;
public class Studente {
   private String nome;  private String cognome; private int matricola;
   private java.util.LinkedList esami = new java.util.LinkedList();

   // il resto del codice della classe resta uguale
```

Media Pesata



le implementazioni cambiano completamente
ma non cambiano le interfacce – l'accoppiamento è minimo

Il Concetto di "interface"

- In sintesi
  - ⇨dato un componente, i componenti che ne utilizzano i servizi dipendono esclusivamente dalla sua interfaccia e non dalla sua implementazione
  - ⇨è possibile cambiare l'implementazione senza influenzare gli altri componenti dell'applicazione
  - ⇨purchè l'interfaccia resti uguale

Il Concetto di "interface"

- Di conseguenza
  - ⇨è possibile vedere la questione dell'incapsulamento da due punti di vista
- Da una parte
  - ⇨un componente è completamente libero di scegliere la sua implementazione
- D'altra parte
  - ⇨è impegnato nei confronti di altri componenti a mantenere inalterata la sua interfaccia

Il Concetto di "interface"

- Nei linguaggi a oggetti
  - ⇨è possibile vincolare una classe di oggetti a fornire una certa interfaccia
  - ⇨attraverso le "interface"
- Interface
  - ⇨strumento per definire l'interfaccia di un componente
  - ⇨elenco di prototipi di metodi che il componente si impegna ad implementare

Un Esempio: java.util.List

```
package java.util;

public interface List {

    boolean isEmpty();
    void clear();
    Object get(int index);
    Object set(int index, Object element);
    boolean add(Object o);
    void add(int index, Object element);
    Object remove(int index);
    int indexOf(Object o);
    int lastIndexOf(Object o);
    int size();
    ...

}
```

Attenzione: si tratta di una versione semplificata della vera java.util.List

NOTA: i metodi elencati nelle interface sono considerati automaticamente pubblici

Il Concetto di "interface"

- Una definizione alternativa
  - ⇨una interface è una collezione di metodi astratti
- Metodo astratto
  - ⇨prototipo di metodo senza corpo
  - ⇨contrario di metodo "concreto": metodo con un corpo e quindi concretamente eseguibile
  - ⇨in Java viene denotato con la parola chiave abstract

Un Esempio: java.util.List

Sintassi alternativa per descrivere l'interface java.util.List

```
package java.util;

public interface List {

   public abstract boolean isEmpty();
   public abstract void clear();
   public abstract Object get(int index);
   public abstract Object set(int index, Object element);
   public abstract boolean add(Object o);
   public abstract void add(int index, Object element);
   public abstract Object remove(int index);
   public abstract int indexOf(Object o);
   public abstract int lastIndexOf(Object o);
   public abstract int size();
   ...

}
```


Il Concetto di "interface"

- Una classe
  - ⇨può impegnarsi ad implementare un'interfaccia
  - ⇨dichiarandolo esplicitamente con la clausola implements
- Esempio
  - ⇨public class ArrayList implements List {...}
  - ⇨public class LinkedList implements List {...}

Il Concetto di "interface"

- In questo caso
  - ⇨la classe si impegna a fare in modo che i suoi oggetti implementino i metodi dell'interfaccia
- Che cosa vuol dire ?
  - ⇨che dovranno, ad esempio, necessariamente fornire un metodo pubblico int size()
  - ⇨non possono non implementarlo
  - ⇨oppure implementarlo con un nome diverso

Il Concetto di "interface"

- Metafora
  - ⇨una interface è un "contratto" tra la classe ed il resto del mondo
  - ⇨ovvero una sorta di "specifica tecnica" sul funzionamento dei robot
  - ⇨la classe si impegna a che gli oggetti che fabbricherà avranno certe caratteristiche
  - ⇨e cioè saranno in grado di effettuare certi compiti

Il Concetto di "interface"

- In questo modo
  - ⇨è possibile garantire la reale sostituibilità tra componenti che implementano la stessa "interface"
  - ⇨es: ArrayList e LinkedList
- Nota
  - ⇨anche le classi possono contenere metodi astratti

Il Concetto di "interface"

- Classe ordinaria (classe "concreta")
  - ⇨contiene solo metodi concreti
- Interface
  - ⇨contiene solo metodi astratti
- Classe astratta
  - ⇨contiene sia metodi concreti che astratti
  - ⇨in una gerarchia, in parte fornisce funzionalità ereditabili, in parte definisce un contratto per le sottoclassi

Riassumendo

- Interfaccia e Implementazione
- Incapsulamento
- Utilizzo delle Liste
  - ⇨La Media Pesata
- Il Concetto di “interface”

Termini della Licenza